1

Shipping Software

WHAT'S IN THIS CHAPTER?

- Understanding what you need to ship software: vision, insight, resources, planning, and product features.
- Understanding three approaches to management methodologies: Scrum, MSF, and Waterfall.
- Comparing the three project management methodologies.

This chapter covers the high-level process of shipping software. You'll learn how to start with a compelling vision and the resources you need to build a product. You'll also learn about the relationship among competing constraints. You'll read about three popular software project management techniques: Scrum, Microsoft Solutions Framework (MSF), and Waterfall. By the end of this chapter, you'll be able to use what you know about the MSF and Waterfall methods to gain insight into Scrum.

This book is about using a specific tool, Visual Studio Team Foundation Server (TFS), to support the Scrum process for shipping great software. The fundamental concept is to put customer value at the center of everything you do. While maximizing customer value, you also maximize your team's productivity and predictability with each release, creating a sustainable team environment for shipping great software. In this book, you'll learn how to use the tools in TFS to manage a software development project using Scrum.

WHAT DO YOU NEED TO SHIP SOFTWARE?

Before you can ship great software, you need to build it. Before you can build it, you need to envision how it might work and look. Before you can envision it, you need to know why and how someone would use it. And before that, you need insight into the problem you're trying

to solve. So, shipping great software really begins with vision and insight into how something works, how it can be improved, and why people care.

This section discusses the following requirements for shipping software:

- Vision
- Insight
- Resources
- Planning
- Product features

Vision

Great software starts with a great vision. It starts with a simple description of what you're setting out to build, for whom, and why. If you can't define this before starting the project, you should think about it some more. Creating software requires a surprising amount of resources, and you need to have a compelling vision in order to attract and allocate the resources you'll need.

The *vision* should be a meaningful description of the product goals and an inspiring outcome. It doesn't need to be melodramatic — helping the world's neediest people or creating the world's newest billionaire — but it must resonate with all stakeholders. The vision should describe the needs and benefits in terms that the user cares about. It should describe the problem being solved and the opportunity in solving it. The vision should pique the interest of customers, sponsors, and the project team.

The product vision should be relatively short; a few paragraphs generally suffice. It should include a description of the problem and opportunity as well as the solution and the benefit. Everyone on the project team will read it. They will talk to their colleagues, friends, and families about it. It should be clear yet comprehensive enough to resonate with these audiences.

The product vision is typically accompanied by a definition of scope. The *scope* clearly sets the path for the vision, from problem to solution. It should include high-level features, time lines, and constraints. You can define the scope of a prototype, a beta, and the first few releases. Earlier milestones should have greater granularity; later milestones can be more vague. The scope should clearly define what is being proposed and what is not being proposed.

Often, the vision and scope are combined into a single document, cleverly called a *vision/scope*. This document is the first opportunity to define a series of releases. Thinking and communicating in terms of successive releases is critical with Scrum. You can define the rough features that will be in each release, so everyone learns to expect incremental progress as the team iterates toward the solution.

For example, say you enjoy fantasy football, a game in which you create virtual football teams that compete based on the statistics of the individual players. The fantasy football sites have great features for building teams and leagues, but they don't combine that with great search capabilities. Using Google or Bing for searching just doesn't give you the data you want. In a moment of inspiration, you decide to fix the problem by building a search engine for this purpose. You could create the following vision for this product:

> *Create a search engine for fantasy sports enthusiasts.*

Based on your vision, the reader will conjure up ideas about what the product may do. It's a good start, but this vision, as stated, is much broader than your intent. Your idea involved fantasy football, and the vision you've created could be interpreted much more broadly. Therefore, you might rewrite the vision as follows:

> *Many websites are available for fantasy football leagues. They have great features but lack powerful search capabilities. We will create a fantasy football search engine so that fans have access to more data and can create the best teams.*

This is better, as it constrains the problem to what you initially had in mind — fantasy football. It also adds a purpose: creating the best teams. Now you need to scope the problem. You want to solve the problem for fantasy football, but you also realize that the solution would work just as well with other sports. So you can broaden the vision and then constrain it with the scope:

> *Many websites are available for fantasy sports leagues. They have great features but lack powerful search capabilities. We will create a fantasy sports search engine so that fans have access to more data and can create the best teams.*
>
> *The immediate market for fantasy sports is 50 dedicated websites, reaching over 3 million users. The indirect market, including news and sports sites, is approximately 500 sites that reach over 20 million people.*
>
> *The first release will target American football and provide an interface for existing websites. It will contain data for all NFL teams and will support parametric searches on player characteristics. Subsequent releases will add more sports leagues; will support features for managing teams, players, standings, and trading; and will have an interface for mobile apps.*

From this vision/scope, the reader will have an idea of the problem being solved, the benefit of the solution, and the iterative path for expanding the product over time. After this, you can define a business case, technical solution concept, and critical assumptions.

There are many examples of vision/scope documents online. They commonly have sections for vision, requirements, solution, assumptions, limitations, and risks. While Scrum doesn't prescribe a vision/scope document prior to initiating a project, it's a best practice to create one in software project management as it forces you to articulate the big picture.

Insight

It's one thing to have a great vision; it's quite another to turn that vision into a great product. This is true of engineering in general and software engineering in particular. Product management has a broad

role in engineering and design. It involves translating the needs and desires of the user or market into instructions for engineering. (We use the term *instructions* very loosely here, as it may take the form of requirements, storyboards, user stories, mockups, visual comps, or other design artifacts.)

As you'll read in Chapter 2, product management is an essential element of Scrum. Say that you're working with a highly skilled, cohesive, experienced engineering team. And say that the working environment allows the team to flourish. This is a great start, but it's insufficient to build a successful product. In this case, the success of the product depends on product management. Specifically, it depends on product management to determine what's needed, by whom, why, and at what cost.

The product owner, who fills the product management role in Scrum, is embedded in the Scrum team and is counted on to know the user intimately. This person must know the user's likes and dislikes, tolerances, and aspirations. The product owner must know what the users love, what they don't like, and what they don't care about. Essentially, the product owner must have insight into how the user will value the product.

As you'll read further in this chapter and Chapters 2 and 3, Scrum is an Agile process for developing software. It prescribes planning and design up front, and it involves more planning and design as you iteratively build the solution. It's far from random and far from chaotic, but it does move fast and allows for unexpected results. Because of this, the product owner's insight into user needs is essential.

Several factors drive the need for insight in the product owner role in Scrum:

- **Decision making —** One person, rather than a group, makes product management decisions. Decisions are made faster than in traditional software creation, but if the product owner lacks sufficient insight into the problem, he or she will make wrong decisions or lack confidence to make any decisions at all.
- **Minimal prototyping —** There is minimal prototyping before the project starts because it's done in early sprints of the project. This means the product owner's insight will quickly be reflected in the product.
- **Iterative nature —** The product features are defined iteratively and frequently. They come in and out of scope based on product owner decisions, which directly impact the value of the product.
- **Customer feedback —** Customer feedback comes early and often. Good insight is necessary to sift out valuable data from extraneous data.

Resources

Building a software product is a very resource-intensive activity. It requires people from many different disciplines working together toward a common goal. These people include visual designers, domain experts, software developers, and product managers. Depending on the domain, building software may also require business analysts, security experts, information architects, and marketing specialists.

It's easy to underestimate the true cost of building production-quality software. When you wrote your first computer program, it was pretty easy: You were the designer, developer, tester, and possibly customer. Indeed, it was a lot of fun, and that's how many of us got hooked on creating software.

Building a commercial product is much more difficult than creating software for yourself. You need to clearly understand what the customer wants before you can start building. You need to know how much the customer is willing to spend before you begin to buy technology or hire the team for the job. Whether you're building something for just one customer or thousands, the work requires significantly more resources than building something for your own use.

As you'll see later in this chapter when we compare different strategies for software project management, Scrum optimizes resources by iterating toward a solution. This enables you to adjust your plans along the way, based on the realities you encounter with technology, the team, and customers.

Time and Money

You need many resources to build and ship a great software product. At the highest level, you need time and money. To be sure, these are not interchangeable, no matter how much you have of either.

The saying "Timing is everything" holds quite true in software development. A perfect product released at the wrong time is generally not very useful. It may be interesting or thought provoking, and it may even be amusing, but if the timing is wrong, the product won't be useful. Similarly, the wrong product at the right time isn't very useful either. It may garner a lot of attention because of its potential, but if the product is lacking some critical element, it won't be very successful. On the other hand, the right product at the right time is very valuable: Even if it has flaws, if you release the right product at the right time, you'll have a success on your hands.

AN EXAMPLE OF A GOOD PRODUCT WITH PERFECT TIMING

One example of the right product at the right time was Microsoft Windows 95. It was a good product with perfect timing. It was the early 1990s, and Microsoft was developing Windows 95 as the next operating system for personal computing, gaming, and business productivity. But a confluence of events occurred during its development cycle that made Microsoft rethink its plan. The Mosaic browser was written and distributed as Netscape Navigator; Senator Al Gore sponsored legislation to increase investments in the publicly funded Internet; and consumer websites such as Amazon, eBay, and Yahoo! began to create real value. Microsoft faced a dilemma: Should it delay Windows 95 until it could include a web browser, or should it ship Windows 95 without one and continue to develop an integrated browser in parallel?

Microsoft knew that people would be upgrading their PCs to get to the Internet, and the company didn't want to miss the opportunity to sell those people an operating system. Microsoft chose the latter route and released Window 95 without a browser but in time to catch the Internet wave. While Windows 95 was far from perfect, its timing was ideal. Windows 95 was included on hundreds of millions of computers, and this cemented Microsoft's place as the desktop of choice for the next decade.

People and Technology

Unlike time, money can easily be converted to other resources you might need. Two obvious resources you need for software development are people and equipment.

For people, you'll likely want a mix of generalists and specialists. You'll need product managers to define the feature set that exactly meets your client's needs. You'll need software architects to stay with the project from beginning to end and to define the technical architecture and design patterns for the product. You may need user interface specialists, database specialists, security experts, and performance-tuning engineers. The bulk of the technical work will likely be done by generalists: engineers skilled in coding and testing.

If your project needs a high percentage of specialists rather than generalists, you might want to increase your time and cost estimates. Finding and replacing specialists can be a more time-consuming and costly activity than swapping generalists.

You can convert money into people by hiring a team. Some team members will be employees — people you want to retain for the duration of the project and beyond. Others will be contractors — people with specialized skills that you need during certain phases of the project.

You'll also need to convert money into technology. You can buy it, lease it, or rent it for remote access. This is true for both hardware and software. You can allocate technology to problems as they arise and then shift resources (such as money) as circumstances change. For instance, you may need significant hardware capacity during a performance-testing phase of a project. In this case, you can acquire the technology on a temporary basis and then shift it elsewhere when you're done.

Planning

Planning is an essential ingredient in shipping great software. Planning takes a great deal of time before a project begins and even more time once a project is under way. This is true whether you're using Scrum or another project management methodology. Depending on which methodology you use, the specific planning activities will vary dramatically.

For instance, planning a project using a traditional software development methodology involves allocating time for requirements definition, design, development, testing, user acceptance, release management, and support. Software development occurs solely during the development phase. Planning a project using Scrum involves allocating time to build features in fixed-duration "sprints." During each sprint, the team conducts the same activities as in the Waterfall method, but it does so in a small set of features. Figure 1-1 shows a high-level view of Waterfall and Scrum planning.

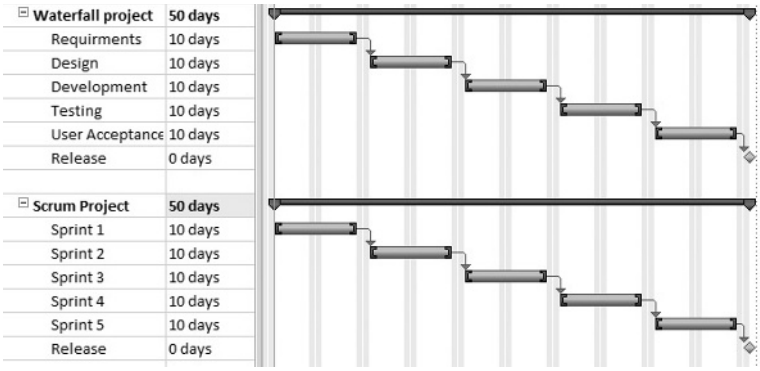


FIGURE 1-1: Waterfall and Scrum planning.

The Waterfall method of project management is the method most commonly used to run software projects. When using the Waterfall method, you schedule tasks sequentially, completing one phase of activity before beginning the next. Later in the chapter, in the "Approaches to Project Management" section, we'll look more closely at the Waterfall method and compare it with Scrum.

One place where Scrum is different from traditional software project management methodologies (such as the Waterfall method) is with respect to predictability. Waterfall assumes that you can predict how long tasks will take. You allocate people and time to tasks and then schedule them accordingly. Scrum assumes the opposite. You cannot accurately predict how long something will take unless you've done it with the same resources (technology and people) before.

If you cannot predict how long a single task will take, how can you predict how long a whole series of tasks will take? Using Scrum, you accept the fact that you can't. Rather than predict the product schedule, you predict smaller units of work that can be completed within a sprint. At the end of each sprint, features are complete and can be included in a potentially shippable product. This way, you predict features by time rather than predicting time by features, which is essentially a time-box approach.

A sprint is the smallest cycle time within Scrum. Chapter 9 focuses on running sprints. Sprints can be as long as a month or two and as short as a day or two. The ScrumMaster decides the duration before the sprint begins. Each sprint starts with a sprint planning meeting, during which the team looks at the product backlog and decides which features to build during the sprint. The team members use their experience in the previous sprint(s) to predict how much they can accomplish in the next sprint. This is significantly different from the Waterfall method, which involves predicting the release and features up front and allocating time and people to the tasks.

Figure 1-2 shows a Gantt chart from a project planned using the Waterfall method. Note that each task is scheduled with a known duration. If a task completes early or late, this will impact all other tasks in the release. This works well if you have a high degree of confidence in your task estimates, but it falls apart quickly if the estimates are incorrect.

Waterfall Project	50 days
Dev Begins	0 days
task 1	3 days
task 2	3 days
task 3	4 days
Release 0.8	0 days
task 4	3 days
task 5	3 days
task 6	4 days
Release 0.9	0 days
task 7	3 days
task 8	3 days
task 9	4 days
Release 1.0	0 days
task 10	3 days
task 11	3 days
task 12	4 days
Release 1.1	0 days
task 13	3 days
task 14	3 days
task 15	4 days
Release 1.2	0 days

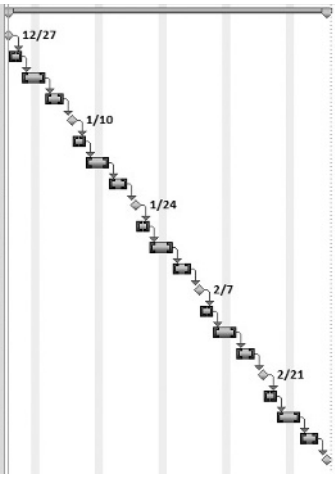
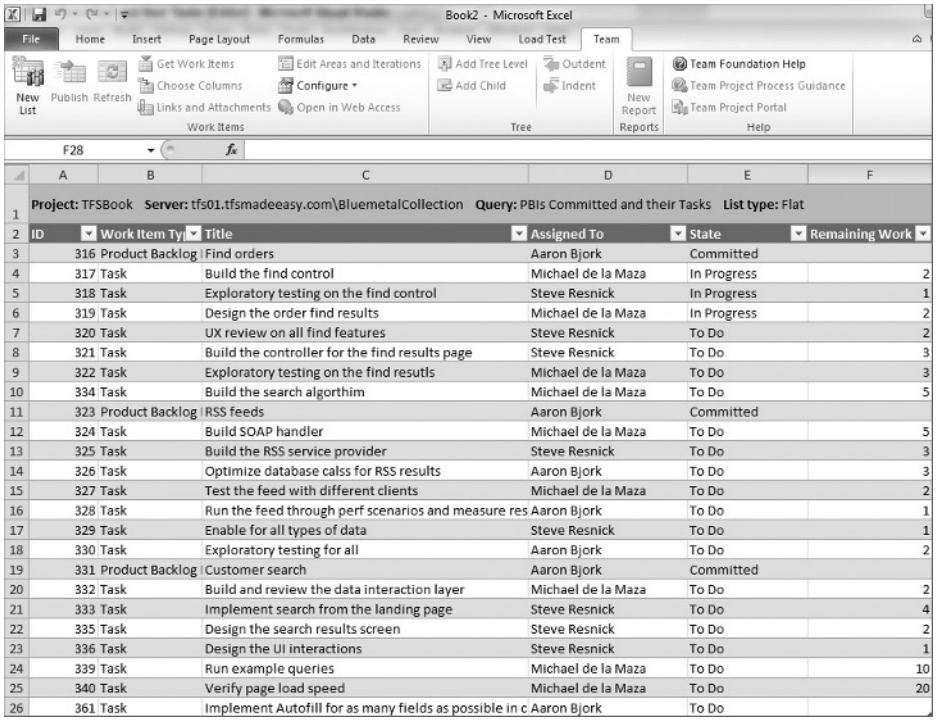


FIGURE 1-2: Waterfall method Gantt chart.

In a Scrum project, you must plan tasks. Software project management requires a lot of planning, and Scrum doesn't change that need. However, rather than plan tasks to manage dependencies, you plan tasks to manage feature delivery. The team focuses on building the product rather than keeping to the schedule. The schedule in Scrum is simple: It's the sprint cycle. Planning within a sprint focuses on the product rather than the schedule because the schedule is so simple. Figure 1-3 shows a project artifact that you'll use for intra-sprint planning. This report, and many others like it, is covered in Chapter 6.



ID	Work Item Ty	Title	Assigned To	State	Remaining Work
316	Product Backlog	Find orders	Aaron Bjork	Committed	
317	Task	Build the find control	Michael de la Maza	In Progress	2
318	Task	Exploratory testing on the find control	Steve Resnick	In Progress	1
319	Task	Design the order find results	Michael de la Maza	In Progress	2
320	Task	UX review on all find features	Steve Resnick	To Do	2
321	Task	Build the controller for the find results page	Steve Resnick	To Do	3
322	Task	Exploratory testing on the find resutls	Michael de la Maza	To Do	3
334	Task	Build the search algorthim	Michael de la Maza	To Do	5
323	Product Backlog	RSS feeds	Aaron Bjork	Committed	
324	Task	Build SOAP handler	Michael de la Maza	To Do	5
325	Task	Build the RSS service provider	Steve Resnick	To Do	3
326	Task	Optimize database calss for RSS results	Aaron Bjork	To Do	3
327	Task	Test the feed with different clients	Michael de la Maza	To Do	2
328	Task	Run the feed through perf scenarios and measure res	Aaron Bjork	To Do	1
329	Task	Enable for all types of data	Steve Resnick	To Do	1
330	Task	Exploratory testing for all	Aaron Bjork	To Do	2
331	Product Backlog	Customer search	Aaron Bjork	Committed	
332	Task	Build and review the data interaction layer	Michael de la Maza	To Do	2
333	Task	Implement search from the landing page	Steve Resnick	To Do	4
335	Task	Design the search results screen	Steve Resnick	To Do	2
336	Task	Design the UI interactions	Steve Resnick	To Do	1
339	Task	Run example queries	Michael de la Maza	To Do	10
340	Task	Verify page load speed	Michael de la Maza	To Do	20
361	Task	Implement Autofill for as many fields as possible in c	Aaron Bjork	To Do	

FIGURE 1-3: A Scrum project artifact.

An artifact *is something that is useful to a project but isn't part of the product itself. Common examples of artifacts are task lists, schedules, and test cases. Figure 1-3 shows a list that combines backlog items and tasks, to easily organize and use them together.*

Budget Planning

It is extremely difficult to accurately forecast the cost of software development before you start. We all wish this weren't the case, but indeed it is. There are many unknowns in the factors that impact cost, such as requirements, technology choices, and team composition. For instance, while you may have a list of high-level or even detailed requirements, you may be two or three steps removed from the source of the requirements when you're preparing the estimate. Or, while you may assume that technology used on a previous project is good for this one, you may get into unfamiliar territory when working on specific features for this new project. And, in the realm of people, you may not be able to assume that a specific person will be on the project team. Rather, you tend to work with roles such as senior architect or database developer when estimating projects.

The project management triangle in Figure 1-4 depicts the three constraints that define a project: cost, time, and features. If you have a full understanding of the features, you'll be able to provide an accurate estimate of the cost and time. This is the case in auto repair and home renovation, so why not with software? The answer has to do with unknowns; in the world of software, you can't know everything before you start. And what you don't know will have to be factored into the triangle, resulting in varying costs, times, and features.

Some people and texts refer to the project management triangle as an "iron triangle" to highlight the fact that you cannot bend it to suit your needs.

Because of these unknowns and the fixed constraints in the triangle, you simply cannot offer a fixed-cost commitment for a fixed-feature product. You therefore have two alternatives: Vary the cost or vary the features. If you must deliver a fixed budget, then the answer is simple: Vary the features.

Scrum offers this alternative. The cost of a Scrum team can be fixed. If you have two product owners, six team members, and one ScrumMaster, then you can add up their weekly wages to determine the weekly cost. If sprints are fixed at four weeks, then you know that a sprint will cost four times the weekly cost of the team. Now you can accurately predict the cost and duration of a sprint. However, customers rarely pay for sprints; they pay for products. This leaves the third element of the triangle: features.

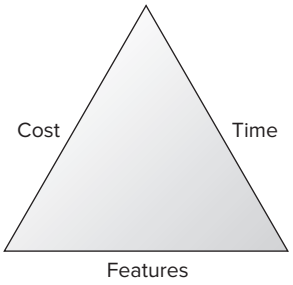


FIGURE 1-4: The project management triangle.

You need a well-developed product backlog. The backlog items must be ranked into a list, based on the value they deliver to the customer. By looking at the prioritized backlog, you can identify the first set of backlog items to implement. You might even be able to identify the second or third set of related backlog items. In other words, by working with the backlog, you can make a reasonable estimate of how much you can get done within a fixed time period. Your estimate is for a fixed time period with a fixed team, but the feature set that you deliver will vary. The Scrum approach therefore meets the goals of budgetary control and also addresses the realities of software development with respect to unknowns.

People Planning

Scrum projects are inherently team-driven activities, with all members involved with all aspects of the project. The team composition is simple, with just 3 roles defined. Team size tends to be small — up to 10 people or so. Because there are so few roles and so few people, team members are highly interdependent. The individuals on the team will succeed or fail together.

Scrum requires a cohesive team, one that is assembled early in the project and remains consistent from sprint to sprint. Of course, people will come and go, as families, careers, and projects take us in unexpected directions, but team consistency is more important in a Scrum project than in other project management methodologies.

In later chapters, we describe how sprint planning begins with a measure of *velocity*, the speed at which a team can implement items on the product backlog. With each sprint, the team estimates the effort needed to complete a set of backlog items. The team commits to completing the items, and then it actually does the work. The team's estimates improve with each sprint, as long as nothing dramatic changes. The measure of velocity becomes more accurate as the team executes sprints.

Changing the team between sprints negatively affects the predictability of the sprints. For instance, if Manny, Moe, and Jack are replaced with Tom, Dick, and Harry, is it fair to assume that they'll complete the same amount of work as the original team? Without knowing how much work these new team members can do, the team will have a difficult time committing to backlog items.

While changing team members between sprints is disruptive and negatively affects productivity and predictability, it's far worse to change team members within a sprint. You should avoid this if at all possible. Changing people will have a ripple effect not only within the team but also potentially within the customer base. For instance, if the product owner told some customers that they can see a feature at the end of a four-week sprint, and team changes within the sprint prevent that feature from being complete, the customer will be dissatisfied.

As with any other process, it's important that participants know who's supposed to do what and when. People do well when they know what's expected of them. The alternative — uncertainty — causes stress. Therefore, you should plan to educate the team on Scrum prior to the project. There are many good books and resources on Scrum methodologies that you can use to help train the team on what to expect. Appendix B provides suggestions on where to begin.

Product Features

Time and money — the latter being converted to people and equipment — are the limited resources for building a product. These resources have a direct relationship to the feature set that ultimately

defines a product. Assuming that there is variability in terms of product features, your job is to optimize the use of time and resources to build a feature set that maximizes the value of the product.

In some business discussions, people are referred to as resources. *This is offensive to some and misleading to others. Here, we refer to resources as just what it sounds like — sources of supply. A resource can be a supply of labor, money, or equipment. Resources can be exchanged, either directly (for example, paying a wage changes money to labor) or indirectly (for example, paying a wage to build equipment changes money to labor to equipment).*

Quality is a feature you can control. You can create a product with more quality or less, depending on how you allocate the resources of time and money. If you run out of time before completing adequate testing on a project, you may implicitly decide to cut quality. The project team has complete control over how much quality to build into a product, as long as quality is considered a feature that requires resources to complete.

Shipping is another feature that you control. You can build a product and not ship it. You may not get paid very much in this case, but it is important to realize that it's a feature. Like quality or other features in the product, shipping takes time and money, so it's important to allocate both. For instance, if you're working on a minor release of a product and you choose to not ship that release, you can re-allocate the time planned for shipping activities ("release" build, final regression test, escrow code, and so on) to other features.

If you have more time or money, you can build more features and quality into a product. Conversely, if you have less time or money, you can either cut features or reduce quality.

The geometry of the project management triangle, with constant angles, dictates the fact that you cannot change just one side of the triangle without adjusting the others. If you increase the features, then you'll have to increase the time and cost lines. If you increase time, then you'll get more features, but it will cost more. If you want to decrease cost, you'll spend less time and get fewer features.

FEATURES VERSUS SCOPE

In general, products have features, and projects have scope. *Features* refer to attributes of the product — such as functions, style, security, or performance. *Scope* refers to the work or effort associated with a project. As constraints on the project management triangle, they're synonymous. But because Scrum is product focused rather than project focused, it's common to think in terms of features rather than scope.

Using Scrum doesn't change the reality of the constraints that define a project. However, Scrum is designed to react to changes gracefully. With respect to scope, Scrum assumes that the feature set is not fully defined up front. Rather, as the product emerges with successive sprints, features are added to and cut from scope. Scrum also assumes that building a product will take a variable amount of time and that you cannot predict the amount of time too early in the project. Once you have achieved a predictable burndown velocity, you can begin to predict scope completion dates. Scrum can work quite well on fixed-budget projects because it enables you to move features in and out of sprints and move sprints in and out of releases.

Burndown velocity *refers to the rate at which you complete product features during each sprint. This is covered in Chapters 8 and 9.*

APPROACHES TO PROJECT MANAGEMENT

This section looks at the three most common approaches to project management:

- **Scrum** — Scrum is the newest approach. Scrum came on the scene around 2000 and is rooted in Agile programming.
- **MSF** — Microsoft Solutions Framework (MSF) was created in the early 1990s. Like Scrum, it is an iterative development method.
- **Waterfall** — The Waterfall method is the most mature process and is firmly established in software and other engineering disciplines.

Scrum

This book is all about using Visual Studio TFS to run a Scrum project. It assumes that you're somewhat familiar with Scrum and are looking for advice and guidance on using the tool to facilitate the process. If you're not familiar with Scrum but know how to use Visual Studio TFS, don't worry. By the time you finish this book, you will know more than enough to begin.

This section presents a very brief summary of Scrum in order to help you compare it with the other software project management methodologies. The subsequent chapters cover specific techniques for using Visual Studio TFS to implement the concepts introduced here. If you're new to Scrum, you might want to check out the many great books, websites, and other resources that we list in Appendix B.

The Theory of Scrum: The Agile Manifesto

The Agile Manifesto is a great starting point for understanding the principles on which Scrum is based. You can find it online at `http://agilemanifesto.org`.

Four high-level values frame the methodology:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation

- Customer collaboration over contract negotiation
- Responding to change over following a plan

These are not values in any moralistic way but preferences for working with products, individuals, teams, and customers on Agile projects. In addition, 12 principles guide Agile software development:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- It's important to welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- It's important to deliver working software frequently, from a couple weeks to a couple months, with a preference to the shorter time scale.
- Businesspeople and developers must work together daily throughout the project.
- An organization should build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity — the art of maximizing the amount of work not done — is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, a team should reflect on how to become more effective and then tune and adjust its behavior accordingly.

The following section describes the high-level process that Scrum follows, from planning through execution.

The Practice of Scrum

Scrum is an *iterative* software development process. In an iterative process, a product undergoes many releases, some major and some minor, with each release adding more value to the product. This type of process enables a team to deliver value to the customer early and to get feedback that can be quickly incorporated into future product development.

In the Scrum method, the product feature set is defined by *user stories*, which are brief narrative descriptions of how the product will be used, by whom, and why. As features are scheduled for development, user stories are decomposed and augmented with increasing levels of detail.

The product release cycle in Scrum is divided into sprints. These are fixed durations, typically two to eight weeks, in which all development activities take place. Each sprint produces a potentially shippable

product that contains features which meet customer expectations. After a number of sprints, typically 3 to 10, the product contains enough value to warrant deployment. Figure 1-5 shows the release and sprint cycle. (Chapters 8 and 9 cover using Visual Studio TFS for release and sprint management.)

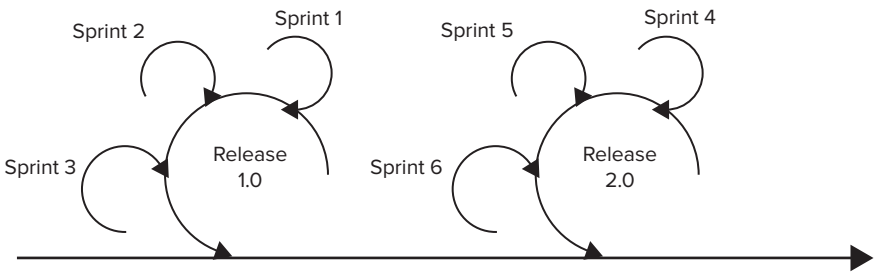


FIGURE 1-5: The Scrum release and sprint cycle.

You can observe the high-level activity of a Scrum project by reviewing three characteristics:

- **Project artifacts —** The artifacts are the lists, charts, and documents that the team uses to run the project.
- **Roles —** The product roles are simply the job descriptions that show who is responsible for what on the team.
- **Ceremonies —** The ceremonies are the rituals that mark the beginning and end of a particular activity.

The following sections look at each of these characteristics in more detail.

Project Artifacts

The product backlog is the list of all features waiting to be built. A team prioritizes the product backlog by business value and ranks it according to which features should be delivered to the customer. Scrum assumes that the list will grow and shrink throughout a release, as the team learns more about the features and the customer learns more about the product. Initially, there is just enough detail associated with each backlog item to begin the discussion between the product owner and the development team. The primary communication mechanism between groups is face-to-face interaction rather than documentation.

The product backlog is the sole input directing work streams of the Scrum team. If a feature isn't on the backlog, it won't be scheduled or built. At the beginning of each sprint, the team moves items from the product backlog to the sprint backlog to indicate the features that will be built in the current sprint. At the end of each sprint, the team produces a potentially shippable product. Bugs that exist at the end of each sprint are added to the product backlog, so work can be scheduled to complete those items in future sprints. (Chapter 6 covers using TFS to manage and track the product backlog.)

Roles

Scrum has a very simple team structure that involves just three roles. This structure generally doesn't translate to an organization or a reporting structure within a company, but it clearly defines who does what on the Scrum team. These are the three roles:

- **Product owner** — The product owner is responsible for all aspects of product definition. This person is the voice of the customer and is always available to meet directly with the development team to discuss and review features. There must be at least one product owner on a project at all times.
- **Team members** — Team members are responsible for building the product. They may follow Agile engineering methods (such as test-driven development or paired programming), although this isn't a requirement. Team members are the architects, developers, and testers. There is no outside group performing these tasks.
- **ScrumMaster** — The ScrumMaster is responsible for the cadence and productivity of the project team. The ScrumMaster defines the sprint duration (generally two to four weeks), runs the daily standup meeting, and helps to keep all team members working productively.

Chapter 2 covers team organization in detail.

Ceremonies

At the beginning of each sprint, the team holds a sprint planning meeting to review the backlog and estimate how much it can accomplish. The team identifies the items it will build in the sprint, and it commits to completing those items. Because each sprint has a fixed set of resources, the number of features must vary (refer to Figure 1-4).

Each day of the sprint, the ScrumMaster leads the daily Scrum, or standup meeting. This is a short (15- to 30-minute) meeting to ensure that everyone on the team is productive and to identify dependences that are impeding progress.

At the beginning of each sprint, the team holds a retrospective in which it looks back and discusses what went well and what didn't, in an attempt to improve productivity. Chapter 10 covers using TFS to conduct effective retrospective meetings.

Microsoft Solutions Framework

MSF is a framework for building and shipping software in an iterative series of releases. Microsoft Consulting Services developed MSF, based on best practices from Microsoft product teams. Its goal is to help a team build and ship software for enterprise customers in a rapidly changing world while reducing risk at each stage. It assumes that there will be changes in scope, technology, and people throughout the project.

Iterative software development focuses on delivering small pieces of functionality frequently in order to solicit and react to feedback, thereby reducing risk. Rather than shipping one release over a two-year project, MSF breaks a release into four smaller projects, each of which delivers a subset of the features. With this method, the end user can see the product in an earlier stage of development and provide feedback before additional features are built.

The MSF Process Model

MSF uses a well-published process model, shown in Figure 1-6. MSF contains five distinct milestones in each iteration, represented in the figure by black diamonds and labeled outside the

circle. There are five corresponding project phases in each iteration. These are labeled on the inside of the circle. The project moves from one phase to the next as each milestone is achieved:

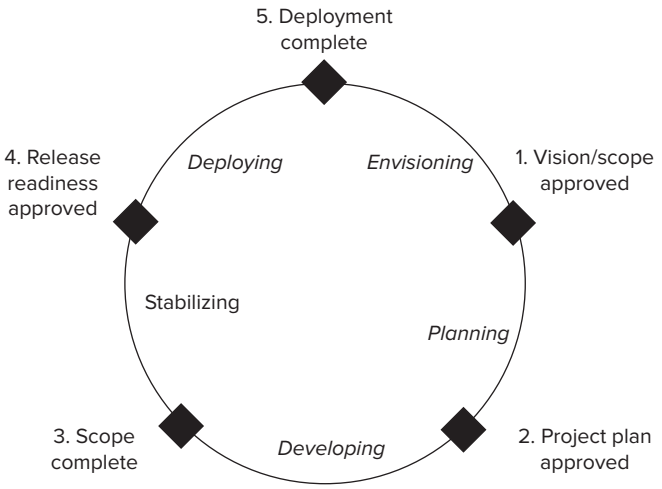


FIGURE 1-6: The MSF process model.

1. **Vision/scope approved** — This milestone is reached at the end of the envisioning phase, after the vision/scope document is reviewed and approved by the project sponsor and user community. In contrast to the Waterfall method, the vision/scope document is typically not an exhaustive list of requirements. Instead, it captures the high-level vision for the release and the specific scope that will be implemented. It may have screen shots of competing systems or of prototypes. It may define high-level use cases, business workflows, or personas to indicate how the system will be used and by whom.
2. **Project plan approved** — This milestone is reached at the end of the planning phase, after the functional spec is written and a concrete project plan is approved by the project sponsor. This is typically a relatively long phase. It involves prototypes and detailed design activities. The more technical work that is completed in this phase, the more accurate project plans will be.
3. **Scope complete** — This milestone is reached after the components are built and unit tested during the developing phase. All major software development is complete at this point.
4. **Release readiness approved** — This milestone occurs after the stabilizing phase, when the system is tested for end-to-end correctness and workflows. This phase also typically includes stress and performance testing. In addition, production-readiness activities, such as run books for operations and configuration tools, are built.
5. **Deployment complete** — This milestone is reached after the deploying phase, when the software is deployed to the target operating environment.

The MSF Team Model

MSF uses a well-published team model, shown in Figure 1-7. MSF defines six roles, all of whom are peers on the project team. Each of these roles should be filled at the beginning of the project, although full-time involvement will vary during each cycle:

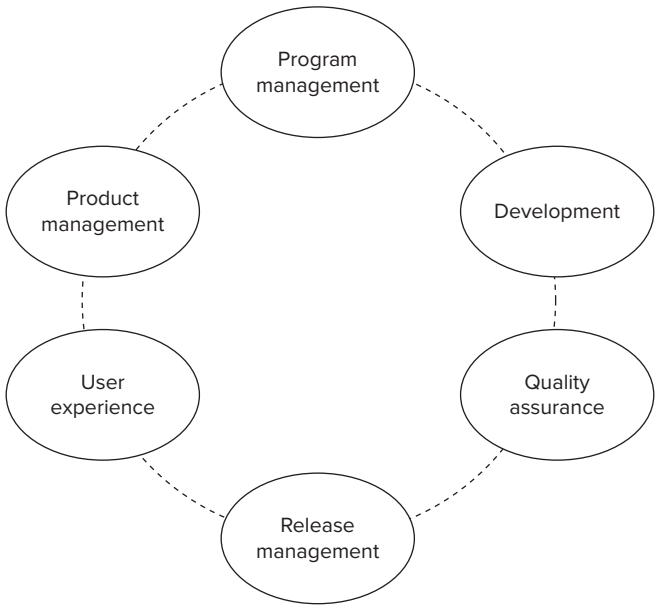


FIGURE 1-7: The MSF team model.

- **Program management —** Program management is responsible for the project plan. Their central job is to balance the project constraints of time and money against the feature set to deliver the product on time and on budget. What makes this difficult is that the program manager does not control the resources or the feature set; the other roles control the resources and feature set. Effective communication and negotiation are hallmarks of this role.
- **Product management —** Product management is responsible for specifying a product that meets customer expectations. Product managers deeply understand customer needs and usage patterns. They instinctively know what is good, what is great, and what is awful. They contribute heavily early in the project, a bit less in the middle, and then significantly again toward the end.
- **Development —** The development team is responsible for the architecture, design, and software construction activities. This team works closely with all the other teams to build a top-quality product. It is organized along functional or technical lines so it can scale well.
- **QA —** The quality assurance team works very closely with the development team and is responsible for tracking and reporting quality to program management. This team develops and executes test plans to ensure that the product meets functional specifications and user

expectations. It works closely with the development team to test the product throughout the developing and stabilizing phases.

- **Release management** — Release management is responsible for the logistics of deploying the product in a target environment. This often includes writing and testing the installation instructions to ensure smooth rollout. It also includes working with operations teams to ensure compliance with local procedures and policies in the target environment. Release management contributes heavily near the end of a release, but earlier involvement greatly increases the likelihood of successful deployment.
- **User experience** — The user experience team is responsible for the overall experience users have with the product. At the software level, it includes visual design, information architecture, feature usability and discoverability, and the overall look and feel of the system. In addition to software, the user experience team delivers documentation, help text, and training. Making this function a peer with other team roles enables these critical functions to be planned for and budgeted throughout the project.

The Waterfall Method

The Waterfall method is a proven technique for engineering and construction management. It breaks a project into a series of phases, each one conducted by a specialized team with specific outcomes and deliverables. The term *Waterfall* refers to the visual structure of a Gantt chart, which is commonly used for planning. Figure 1-2, earlier in this chapter, depicts the waterfall shape of the Gantt chart.

The Waterfall method is very effective under certain circumstances, although it has had limited success in producing modern software. It's used in situations in which there are well-understood requirements and the solution uses proven, mature technology. The Waterfall method favors stability over agility, planning over experimentation, and documentation over discussion. The following sections discuss these three concepts.

Stability

If system requirements are stable, you can predictably engineer a solution that meets those requirements. For example, if you're hired to build a bridge across a river, you will be given some very concrete requirements. You will be told where the bridge should begin and end. You'll be told the volume and makeup of traffic it must carry. There are dozens of other requirements, but for the sake of this example, let's ignore them and assume that they are relatively predictable. With stable requirements and stable technology, an experienced engineering firm should be able to prepare a reliable estimate for completing the work.

If the requirements are more dynamic, then it becomes increasingly difficult to estimate the time and cost of the project, and the Waterfall method fails. For instance, if you are hired to "move people between Boston and Cambridge as efficiently as possible," you cannot predict when you will be finished because you don't yet know if you'll be carrying foot traffic, cars, or trains and whether one bridge is more desirable than two. In this case, an experienced engineering firm would propose a discovery phase, possibly for a fixed price, but could not accurately plan the project further.

With software, customers typically describe the solution they want ("move people") rather than the product they want ("a bridge"). This makes using the Waterfall method difficult because there are simply too many unknowns at the start of a project.

Don't underestimate the language barrier between customers and engineers. Customers may have difficulty articulating what they want, and engineers may not fully understand what they need to build. If these people are speaking different languages, each will be misunderstood.

Planning

Each phase in the Waterfall method is predicated on successfully completing the prior phase. Each phase builds on the work and decisions made in the prior phase, and the team can adjust its plans accordingly. This implies that decisions made in early phases have an increasingly large impact in later phases. That being the case, earlier phases focus on solidifying the requirements and design of the system, with little code or engineering work taking place until later.

A central concept in this method is that problems uncovered earlier in the process are much easier to correct than those found later. For instance, in the bridge example, it would be very expensive to move the bridge once constructed, so the engineering team has to be 100% sure of the location before the first stone is moved. The same concept can also be applied in some software projects. If the requirements are very stable and clearly understood, then a lengthy design phase, including prototyping and feedback, followed by a shorter development phase, can deliver a product for a predictable cost.

Documentation

Documentation is the primary communication vehicle between phases of the Waterfall method. Heavy reliance on documentation allows project phases to start, be staffed by experts, produce a result (a document), and then wind down in a predictable manner. It also enables a large team to switch players throughout a project in order to maximize people's time. Finally, it provides a written record of progress so there is transparency into why, when, and by whom decisions are made.

Requirements gathered from stakeholders are cataloged and assembled into a document that becomes the definition of success. This primary document is called the requirements definition, business requirements document, or something similar. The document essentially becomes the contract between the business users and the technical implementation team. If, when the system is deployed, it meets the requirements listed in this document, the system is deemed successful. Therefore, both teams must fully understand this document.

After the requirements definition is reviewed and approved by the business users, a more technical team translates it into a functional specification (or spec). The functional spec describes what the system will do. It depicts screens, database tables, field-level validation, and workflow. It translates the business requirements into something that the team can build. The business user must also review and approve this document, since it precisely describes the system that will be built. The functional spec also typically includes a traceability matrix that references the requirements document. This matrix ensures that the functional specification addresses all business requirements.

Following the functional spec is a detailed design document, the first document that addresses the technology. Its purpose is to map the functional spec into a blueprint of a system. After system architects approve the detailed design, construction begins.

While heavy documentation has advantages in terms of oversight and traceability, there are major problems with it. First, it assumes that people will read the documents. This is rarely the case, as documents frequently exceed hundreds of pages. Second, it assumes that the reader can understand the documents. This is also rarely the case. Business users don't speak or write in terms of "requirements," yet they are expected to approve a document written in that language. They generally approve a document based on their trust of the people writing it, but this allows errors and omission to easily slip past review. Finally, the more the team focuses on documentation, the less it focuses on the actual task at hand — building great software.

COMPARING METHODOLOGIES

The following sections compare the three project management techniques according to primary characteristics. The intent is to help you understand the similarities and differences, so you can use your experience with Waterfall or MSF to gain insight into Scrum.

Product Definition

How, when, and by whom is a product defined? How are the user needs and expectations captured? How are requirements communicated to the technical team? How does the technical team communicate with users? The three project management methodologies address product definition as follows:

- **Waterfall** — Product requirements are extensively documented during the first phase of a project. They are generally expressed in a technical grammar rather than natural language, although they are describing business goals rather than technology. At the end of the requirements definition phase, the capabilities of the system are fully specified. The requirements document can be used for tendering proposals from competing vendors to do the implementation.
- **MSF** — The product definition begins with a vision/scope document, which is a narrative description of the high-level goals and motivations of the project. This document is used to build a functional specification that fully describes the product. The functional specification can be used for tendering proposals from competing vendors to do the implementation.
- **Scrum** — The product definition is captured as user stories and expressed in natural language, in the form `<someone> wants to do <something> because <reason>`. User stories are decomposed and expanded closer to implementation. The feature set for the system is dynamic and changes throughout the project life cycle.

Adaptability

How does each methodology work with changing requirements? Does the methodology favor a stable or changing landscape? The three project management methodologies address adaptability as follows:

- **Waterfall** — Requirements are locked down early in the project life cycle. Changes introduced later in the project can have a large ripple effect on time and cost. Change orders are used to track and schedule cost and features. A big design phase up front can produce a predictable cost and schedule.

- **MSF** — MSF features iterative development that reacts well to change. Requirements are locked at the beginning of a release but can be added in subsequent releases. Major and minor releases can be scheduled based on new requirements.
- **Scrum** — Scrum assumes that features will be added to the product backlog after work begins. Because change is expected, it has less of a ripple effect throughout the system. Instead of change orders, additional sprints or releases are added to the schedule to implement new features.

Scheduling

What is the basis for scheduling features or people? When will you know if the project is slipping? The three project management methodologies address scheduling as follows:

- **Waterfall** — Scheduling is predictive. Using a known team and known technology, an experienced team can predict the duration of each phase and task. This method doesn't respond well to slippage, as dependencies among tasks and phases are often very complex.
- **MSF** — Scheduling is predictive, as in the Waterfall method. However, because MSF is iterative, with more frequent releases, schedule slippage is more manageable. Subsequent releases can add or remove features to react to prior impact.
- **Scrum** — Scheduling is empirical. Work is scheduled based on the Scrum team's velocity. Estimation becomes more accurate with each successive sprint, based on actual work completed. Scheduling is very reliable because of the fixed-duration sprints. The scope is less reliable because features will move in and out of sprints and releases to accommodate the fixed schedule.

People

How are teams organized? When are people added to and removed from projects? The three project management methodologies address people as follows:

- **Waterfall** — Specialized teams work on different phases of the project. Business analysts perform the requirements definition work early, often before technical experts have been assigned to the project. Once development begins, business analysts have a reduced role. Testing begins after development completes. Project management is a specialized role, often staffed by a project management office.
- **MSF** — Specialized teams work on different aspects of the project but all at the same time. Roles are clearly defined and cover the full spectrum of what's necessary to build and ship products in a predicable manner. The project is run by a team of peers, with each discipline contributing to each phase.
- **Scrum** — A single team is involved throughout the project life cycle. Within the team, just three roles are defined. The work is very collaborative within and across roles. The team is self-organizing, and team members have full visibility into the product backlog and commit to completing scope. The team is involved in planning, estimating, developing, and testing. The team remains customer focused throughout the project.

Documentation

What form of documentation is needed and produced? The three project management methodologies address documentation as follows:

- **Waterfall** — Documentation is the rule of law. Documents describe what's needed and how the system will work. Documents enable team members to come and go because they provide a permanent record of decisions. Microsoft Project and Gantt charts are tools commonly used for documenting the project schedule.
- **MSF** — A prescribed set of documents guide an MSF project. Beginning with a vision/scope and concluding with release documentation and the Microsoft Operations Framework, these documents provide a common language for teams familiar with MSF. Because MSF is primarily used on Microsoft-focused projects, documents are frequently stored in SharePoint or Visual Studio. Microsoft Project and Gantt charts are tools commonly used for documenting the project schedule.
- **Scrum** — Discussion and informal communication are favored over formal documentation. User stories are decomposed into a scope that is scheduled for development. Before work can begin, the product owner and team members discuss the features in detail. Visual Studio TFS is a very effective tool for communicating user stories, features, and tasks. When using Visual Studio TFS for Scrum artifacts and activities, documents are frequently stored in SharePoint.

Project Duration

What is the typical duration of a project? The three project management methodologies address project duration as follows:

- **Waterfall** — Waterfall typically involves longer development projects, often measured in years. It's not uncommon to spend 3–4 months defining business requirements, followed by 3–4 months defining the functional requirements and then 3–4 months defining the technical design, all before the software development phase.
- **MSF** — MSF uses an iterative framework, with releases shorter than those in Waterfall projects. Typical durations are 6–12 months for a major release and 3–6 months for a minor release. This pace balances design with delivery and user feedback with product improvement.
- **Scrum** — Scrum excels with projects of variable lengths and scopes, especially those that deliver value to the customer early, with rapid iteration and product improvement. Releases typically last 6–12 months, and sprints last 2–4 weeks.

SUMMARY

Shipping great software requires a lot more than writing great code. It requires all of the following:

- **Vision** — A great product starts with a compelling, concise description about what you're building, for whom, and why. This is written or heavily shaped by the project sponsor.

- **Insight** — Shipping great software requires a deep understanding of the desires, needs, and tastes of the customers. It is the product owner's job to have this understanding.
- **Resources** — It takes a surprising amount of resources — including time, money, people, and technology — to build and ship software. You need to allocate and spend resources carefully.
- **Planning** — For planning, you need a process, and you need tools. Scrum is a planning process, and TFS is a planning tool.
- **Features** — Ultimately, the success of your software is measured by the usefulness and quality of what you produce. It can be considered useful in terms of productivity, education, entertainment, or any other attribute your customers value. Scrum is a product-focused method for producing products with the right features for your customers.

With all this in place, you have the raw ingredients to ship great software.

Several project management methodologies are commonly used in shipping software. This chapter discusses three of them: Scrum, MSF, and Waterfall. It presents highlights of the three methodologies and compares their significant project attributes.

You're now ready to begin learning more about running Scrum projects. In Chapter 2, you'll learn about the organization of a Scrum team.